Exploiting Computational Slack in Protocol Grammars

Meredith L. Patterson
Independent Researcher

Len Sassaman
K.U. Leuven ESAT-COSIC / The Shmoo Group

PH-Neutral 0x7da
29 May 2010

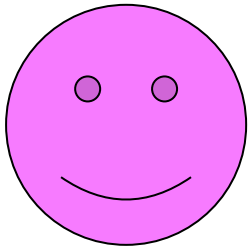
What This Talk Is Not

- A tool
  - Yet
- A library
- A framework
- An all-in-one security solution

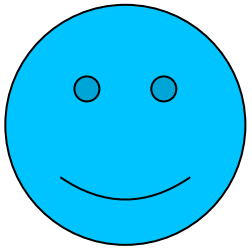
But it is…

- A methodology for implementing protocols
- A methodology for attacking protocols
- A new paradigm for thinking about protocol design
  - …with some very old roots

The Classic Communications Model

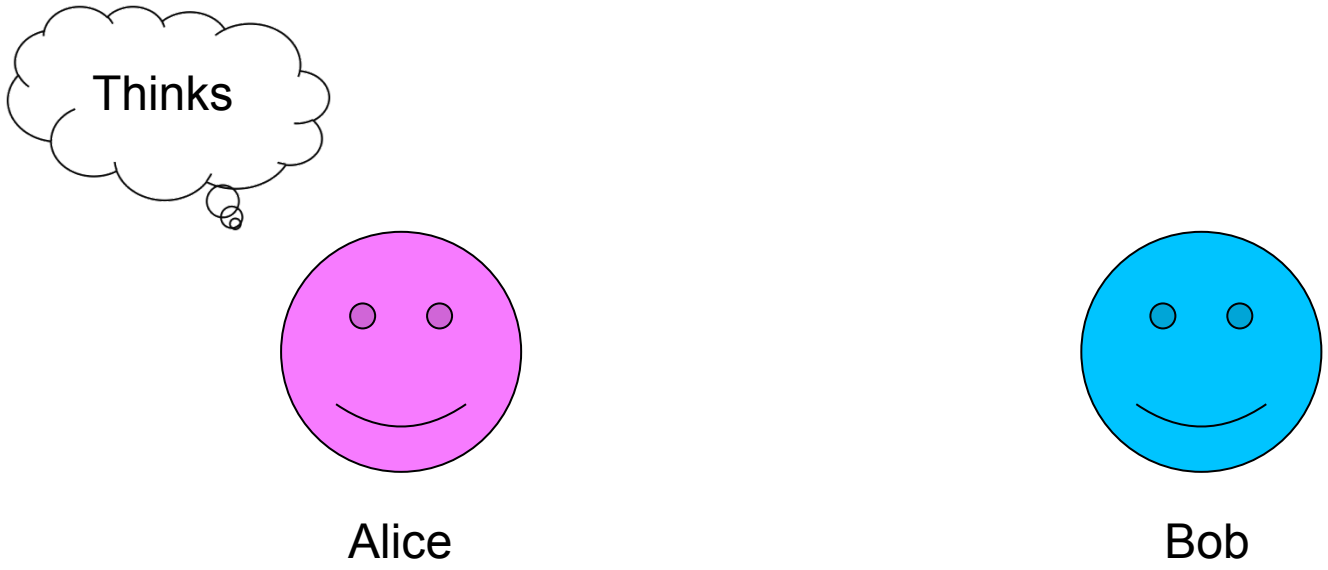


Alice

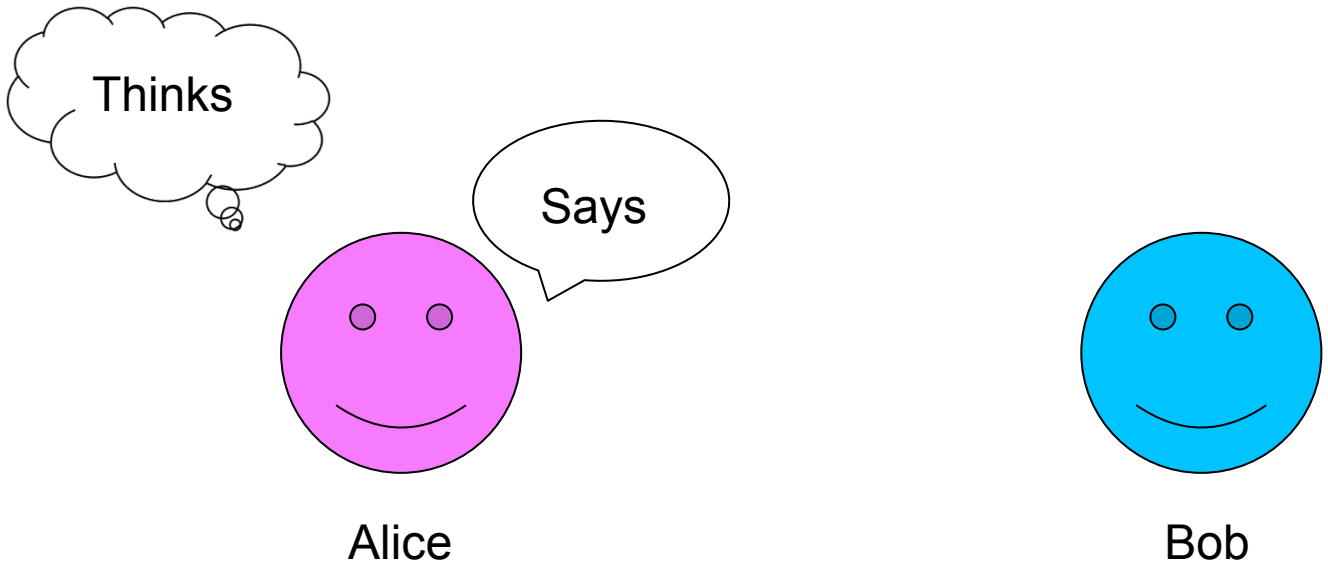


Bob

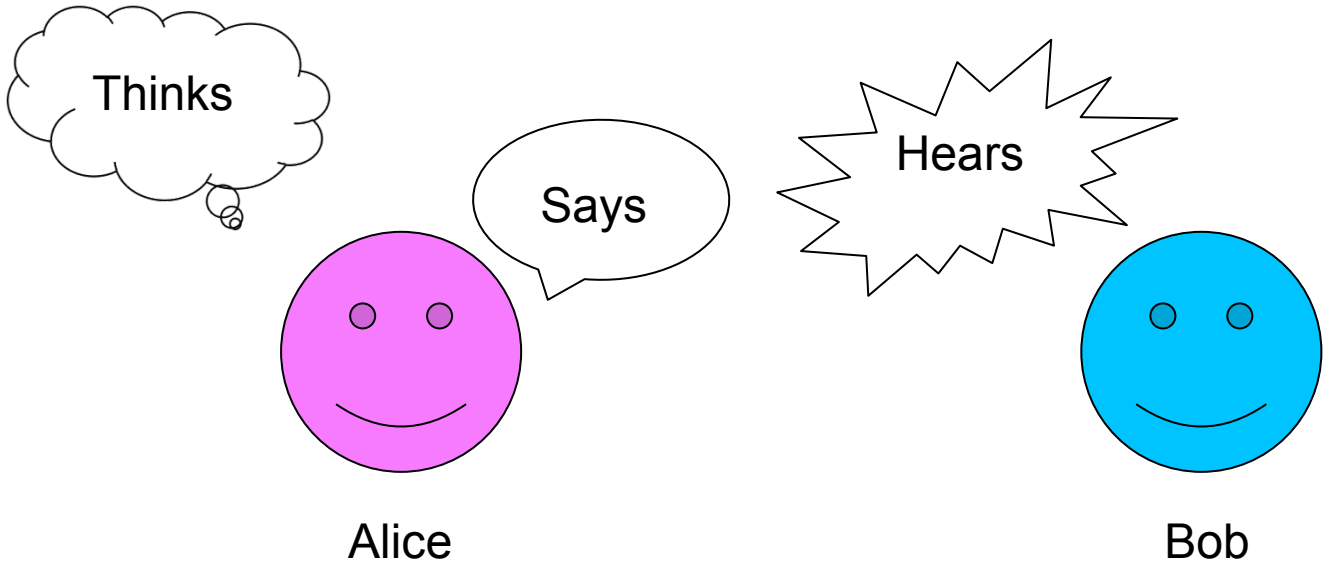
The Classic Communications Model



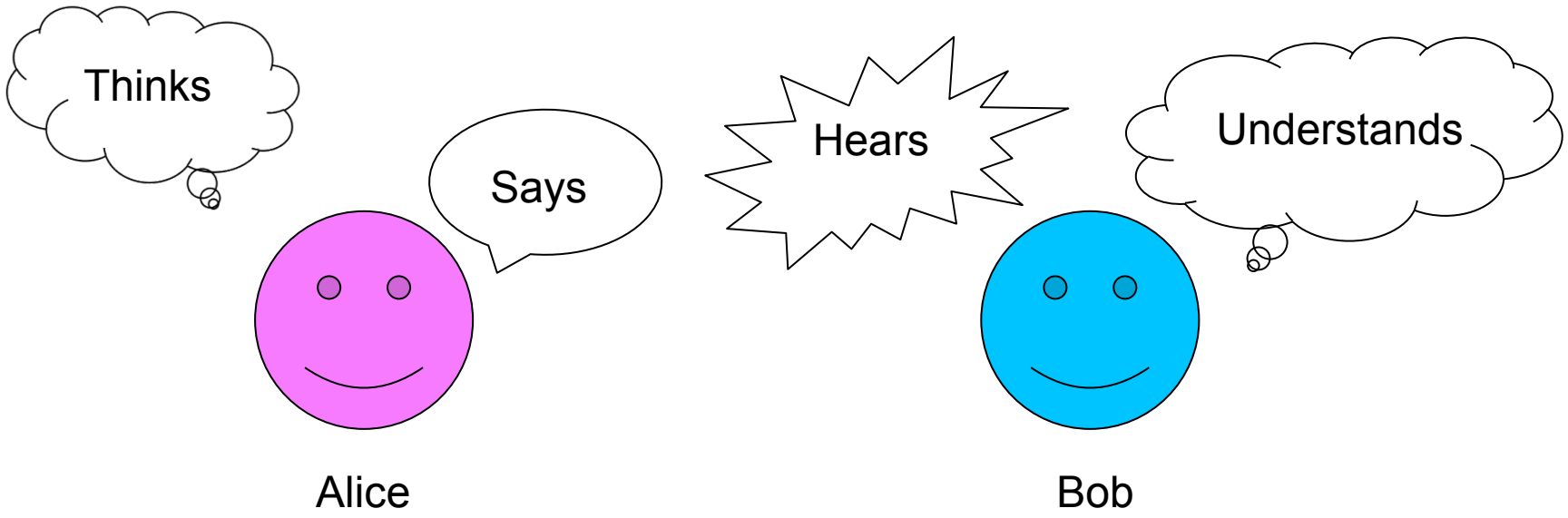
The Classic Communications Model



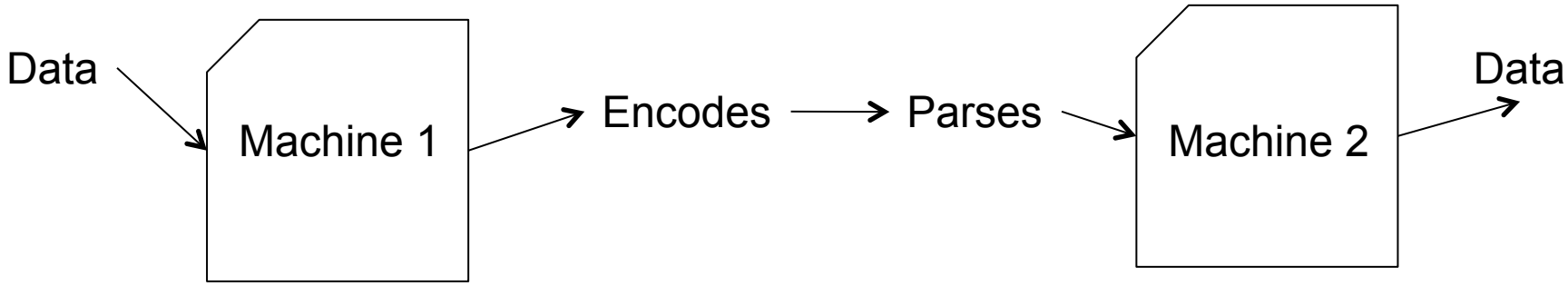
The Classic Communications Model



The Classic Communications Model



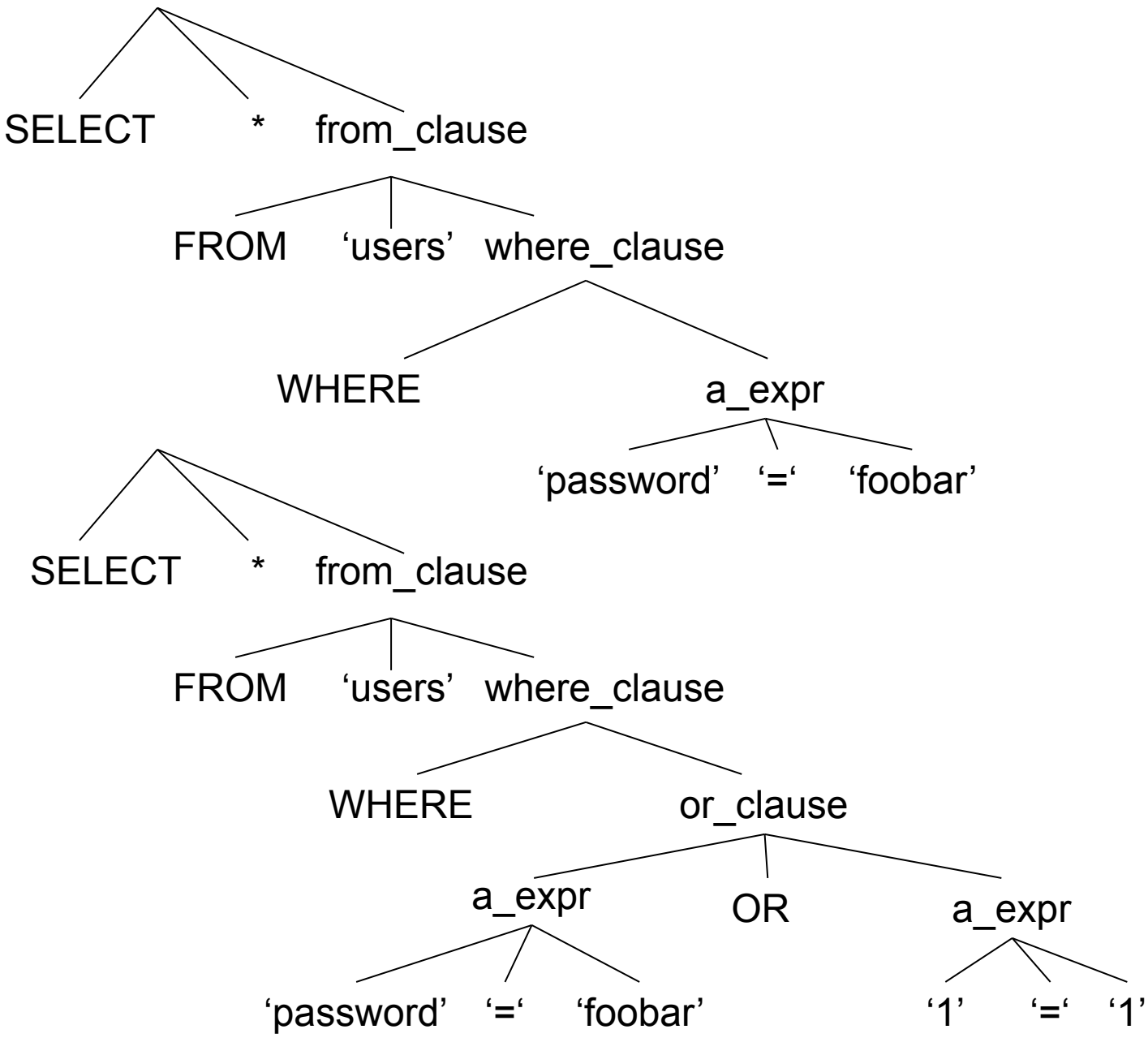
…applied to data



Keep this in mind.

Background

- 2005: Dejector
- SQL injection = mismatch between expected and input parse trees (regex/whitelist fail)
- Compare parse tree of input string with parse tree of expected “exemplar” string
- Differing nodes in input parse tree indicate clauses injected by attacker
    - Malicious OR clause (“OR 1=1”)
    - Malicious commands (“; DROP DATABASE; --”)


SELECT
*
from_clause
FROM
‘users’
where_clause
WHERE
a_expr
‘password’
‘=‘
‘foobar’
SELECT
*
from_clause
FROM
‘users’
where_clause
WHERE
or_clause
a_expr
OR
a_expr
‘password’
‘=‘
‘foobar’
‘1’
‘=‘
‘1’

Lessons from Dejector

- Trying to validate strings in a context-free language using regular expressions doesn't work
- More generally: you must validate with a computational mechanism at least as strong as the language being validated (minimum validation strength)
- Syntactic differences produce semantic differences
  - Parsers are the gateway to code execution

From context-free to context-sensitive

- 2006 CRYPTO rump session: Bleichenbacher PKCS#1 padding attack
  - Signatures padded to the width of the modulus
  - Number of padding bytes varies as a function of hash algorithm and modulus
  - Decrease the number of padding bytes, add unchecked garbage after hash, forge signature for e=3 (or 5?) [$2^{16}+1$ is plenty safe]
- PKCS#1 is context-sensitive, so parse tree comparison alone doesn't work
  - But an attribute grammar comparison does

The Classic Communications Model

- What's said: A signature that is strictly ungrammatical
  - Signature is too long, contains nonsense that shouldn't parse
  - Padding bytes are too short
- What's heard: A string with the correct overall length, possibly including some number of padding bits
- What's understood: A valid signature

Semantic Distinctions

- What we learned from Bleichenbacher:
  - If you rely on an implementation that can interpret an invalid string as valid, you lose.
  - We can validate context-sensitive languages
- Scripting languages, message formats, file formats, network protocols – all formal languages.
  - ...more or less. (ASN.1, ECMAscript = less.)
- Each implementation is its own dialect.
  - “Mutually intelligible,” in linguistics terms.

Deconstructing the Attack Surface

- What the sender means and what the recipient interprets are not necessarily the same thing
- Parsing variations lead to ambiguous meaning
  - Implementation A intends for a string to represent BIGINT, but implementation B parses it as int32
  - Implementation A deletes null bytes, but implementation B parses them as terminators
- Users don't notice what they don't do themselves
  - Implementation-to-implementation communication is more important than user-to-implementation communication!

Methodology

- Consider the formal grammar (as far as possible)
- Examine implementation details
  - How do bytes become data?
  - How does data become bytes?
  - How are unexpected symbols handled?
  - Does this code really implement the grammar as declared in the spec?
- Compare implementations at points of variance

Case Study: X.509

- Parse-tree differentials revealed multiple attack vectors:
  - Overflows (buffer *and* integer!)
  - Spurious NULL
  - Extra padding
  - Injection
- Parsing differences in both browsers, web servers, and CA software.

Subject Name Confusion

- X.509 Name = ASN.1 Sequence of Sets of Sequences of OID/String pairs
- Common Name only relevant one for browsers
  - Name of site being secured is compared against CN
  - If CA doesn't validate CN correctly, it will issue certs for names the user doesn't own

Multiple Common Names

- If a CN contains more than one Sequence of OID/String pairs, which one do we assume is the CN?
  - Spec doesn't say, formally.
  - OpenSSL: “Here, have a list of all of them”
  - CryptoAPI: “Here, have a list of all Strings matching OID 2.4.5.3”
  - NSS: “Here, have the last element of the sequence.”

Inefficient BER Encodings

- BER is complicated and lax, leading to multiple possible encodings for OIDs
- Leading-zero padding:
  - OpenSSL catches 2.5.4.03 internally but presents 2.5.4.3 textually
  - CryptoAPI: 2.5.4.[0]*3 == CN!
- Integer overflow:
  - CryptoAPI: $2.5.4.(2^{64}+3)$ == CN!
- Integer underflow?

Early NULL Termination

- ASN.1 strings are Pascal-style, not C-style
- CN=www.bank.com[NULL].badguy.com:
  - OpenSSL: www.bank.com\x00.badguy.com
  - Crypt::OpenSSL::X509: www.bank.com.badguy.com
  - OpenSSL X509_NAME_get_text_by_NID: www.bank.com
  - Firefox/IE: www.bank.com
- Moxie Marlinspike discovered this specific attack as well, through intuition.

OpenSSL compat mode injection

- OpenSSL can emit CNs as text from command-line
- Other scripts use this data, but it's ASCII
- Throw an escape character into a non-validated Name, e.g. organizationName:
    - ON=Badguy Inc/CN=www.badguy.com
    - Oh, look, the first CN we regex out is www.badguy.com, must be ok
- If you use command-line OpenSSL, use the nameopt flag.

PKCS#10-tunneled SQL injection

- In ASN.1 there are many types of String
  - BMPString: UTF-16, sort of
  - UTF8String
  - UniversalString
- Embed these in a PKCS#10 request, and you can embed any Unicode SQL injection attack you want

PKCS#10-tunneled ASN.1 attacks

- In 2002, PROTOS finds many, many ASN.1 vulnerabilities by way of SNMP
    - ... especially in BER
- Commercial CAs: was your ASN.1 BER parser covered?
    - We did not “test CAs' parsers for them,” that isn't nice.
- BER is so permissive, PROTOS might not have gotten everything
    - DER is much less complicated, and safer

Lessons Learned

- Language Theoretic Security is not just a defensive tool.
- Differential parse tree attacks are quite powerful
- The more ambiguity in a spec, the more attack vectors will be available
  - Yeah, that means you, ECMAScript and IPv6
- Cryptography is rarely the weakest link (again)

Security at the Language Level

- Postel's Law is outdated
  - Conservativeness in what you send is good
  - Liberalness in what you accept exposes you to attack
  - Attacker only needs to figure out what he can make you *think* he said
- “Mutually intelligible” dialects invariably lead to exploits

Context-free equivalence problem: a spanner in the works

- Determining whether two context-free (or stronger) grammars generate the same language is UNDECIDABLE
  - Except in the case where the rules and symbols of grammar G are a subset of those in grammar H
  - We proved that way back in 2005 (Dejector)
- Implementations that don't generate parsers from the specified grammar cannot be guaranteed to implement the grammar's language!

Grand plans

- Language validation shims at strategic locations
  - Network stack, kernel, hypervisor
- Protocols to be implemented with a reference grammar
- Canonical reference grammar specs, machine and human readable, used by the parser to validate different protocols dynamically
  - Protocols can be wire, file format, code execution format, etc.

Questions?

mlp@thesmartpolitenerd.com

len.sassaman@esat.kuleuven.be